

Fig. 1 (prior art)

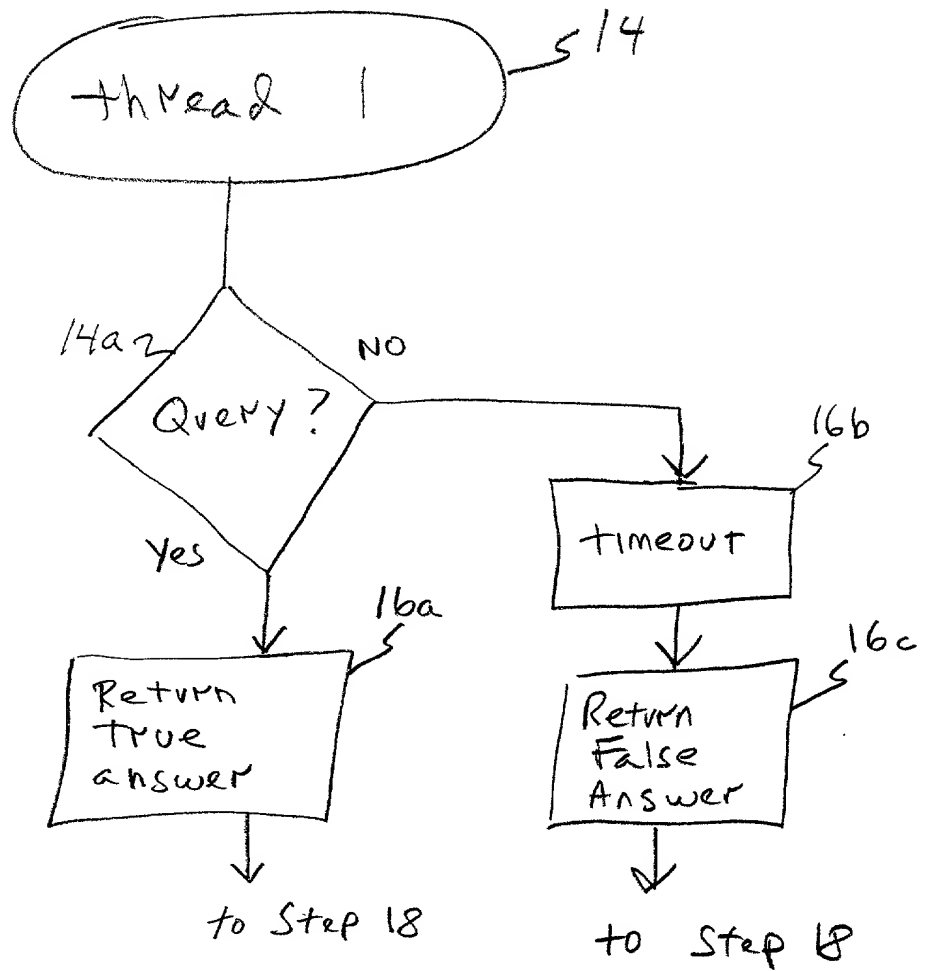


Fig. 2 (prior art)

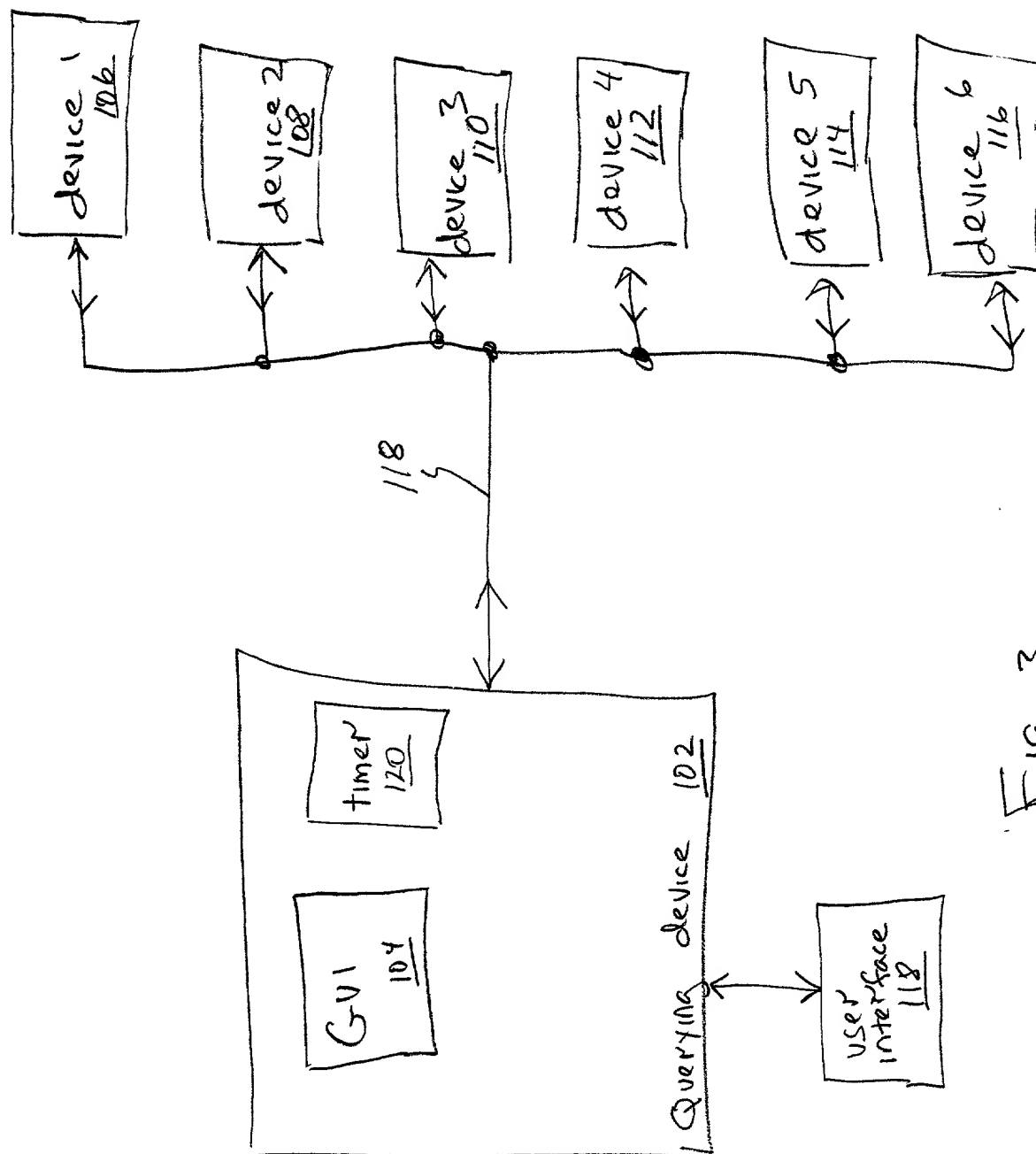
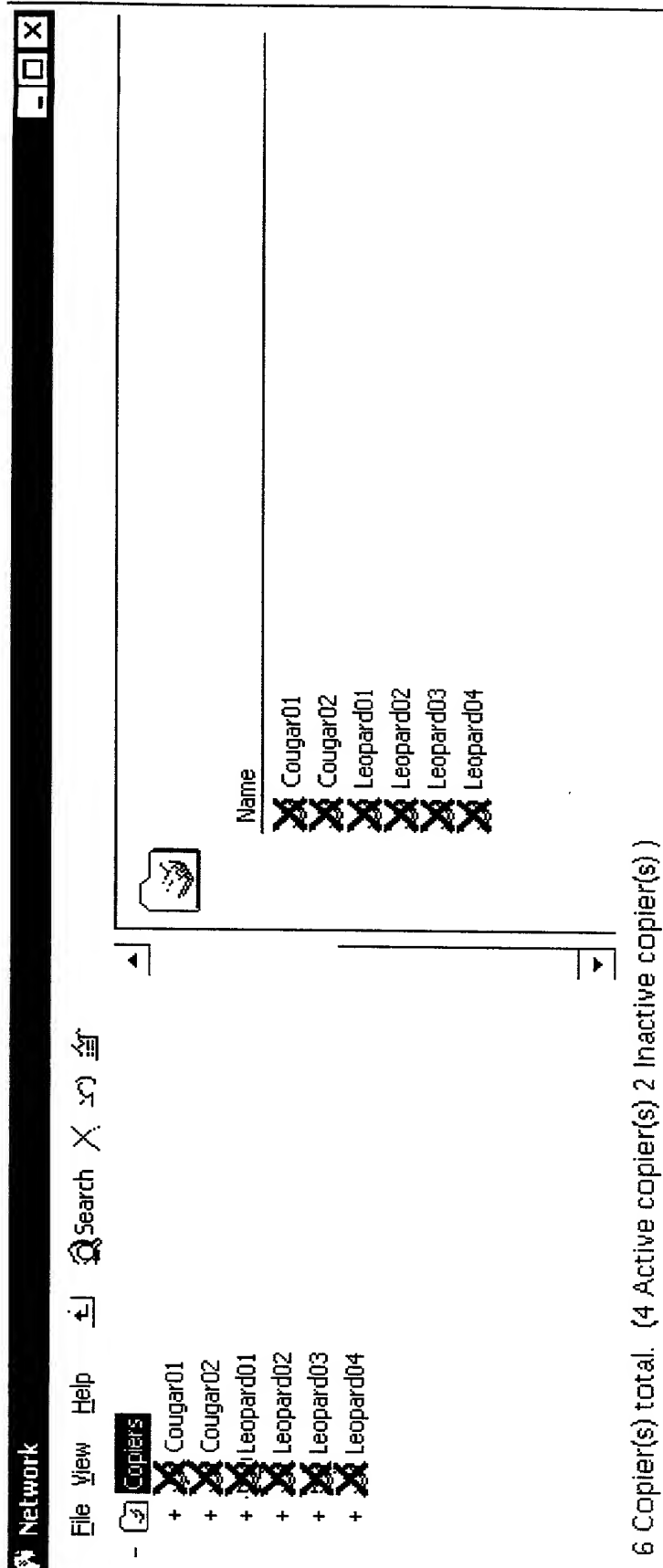


Fig. 3



4104

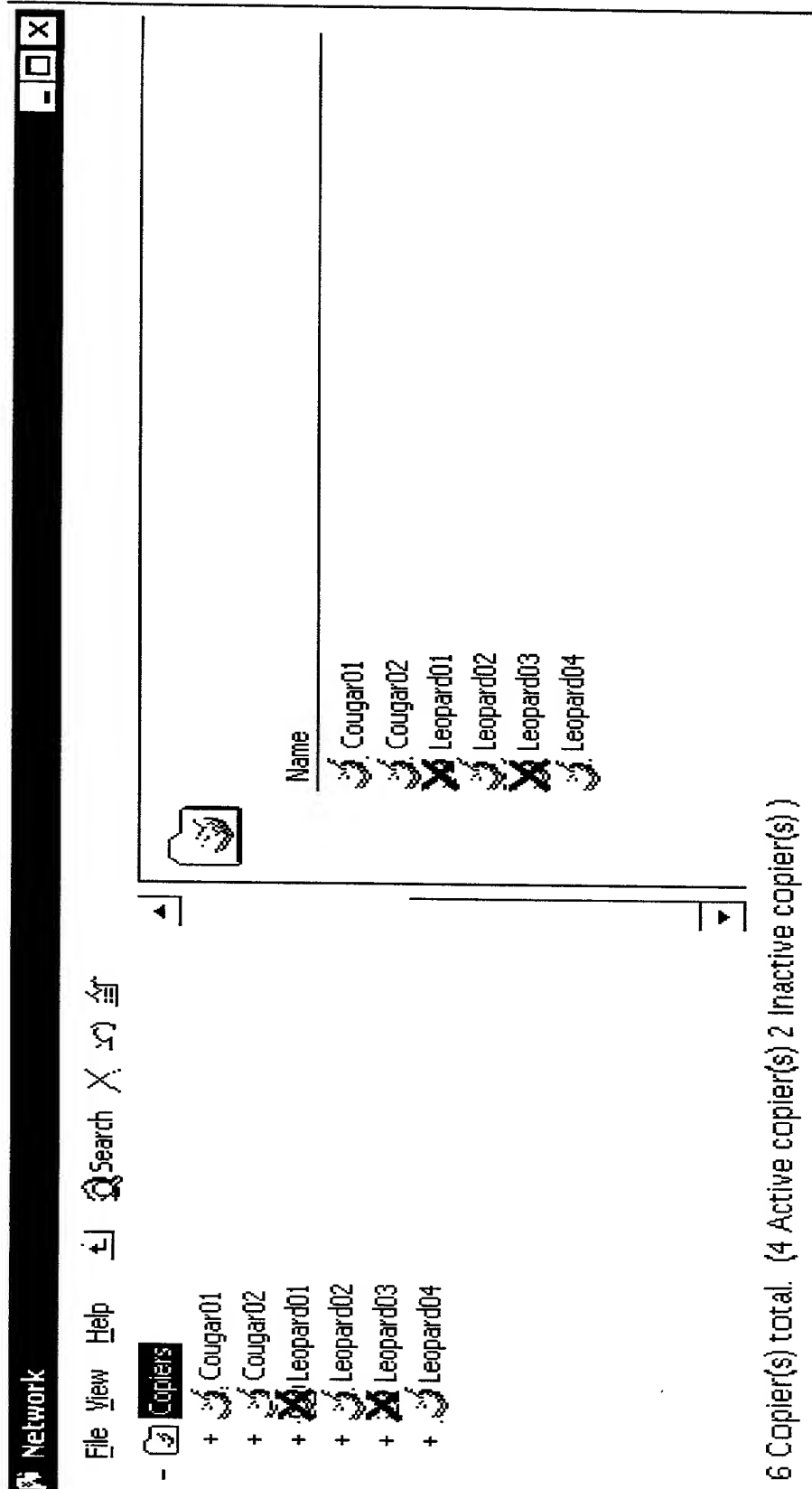


Fig. 5

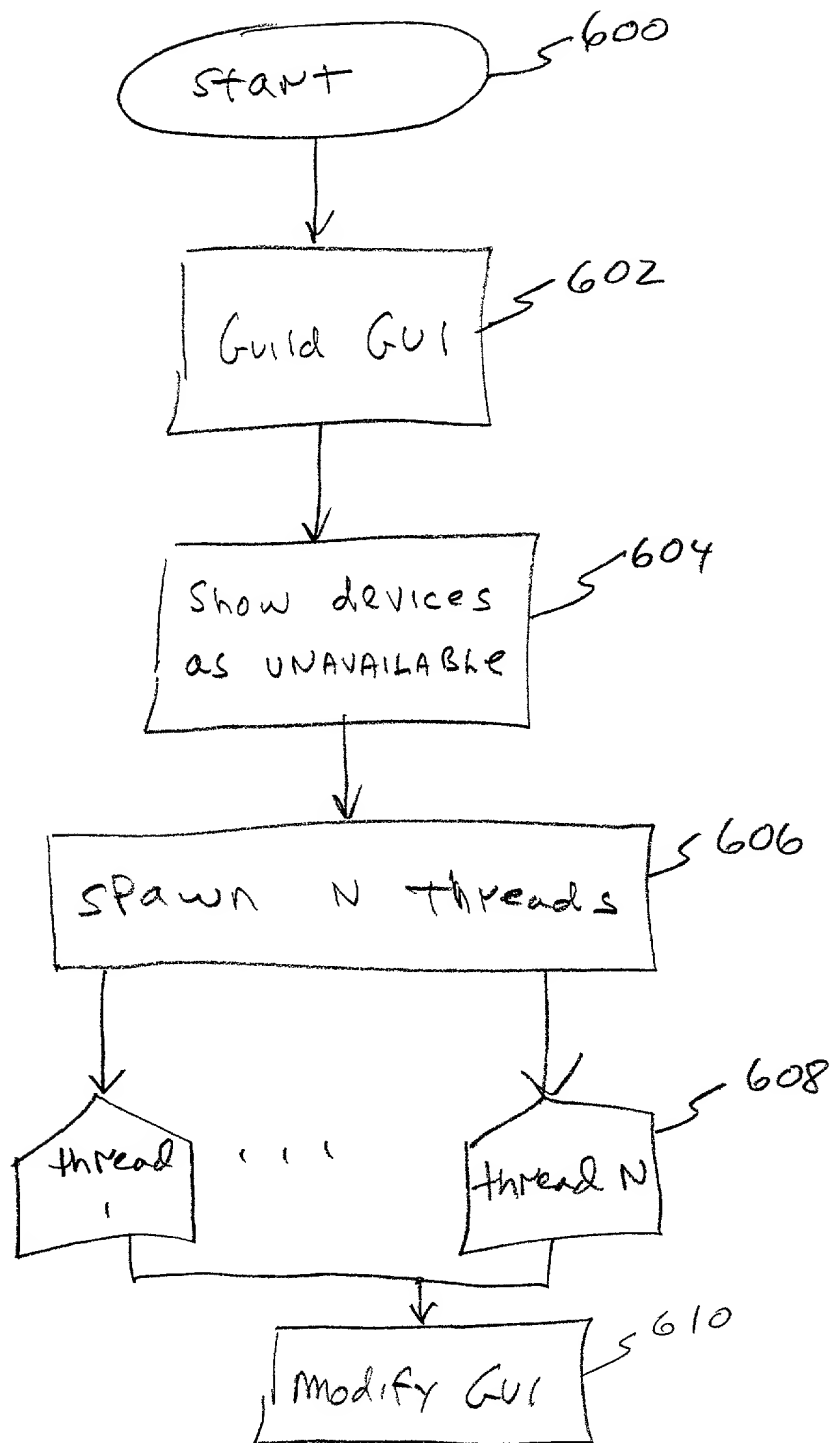


Fig. 6

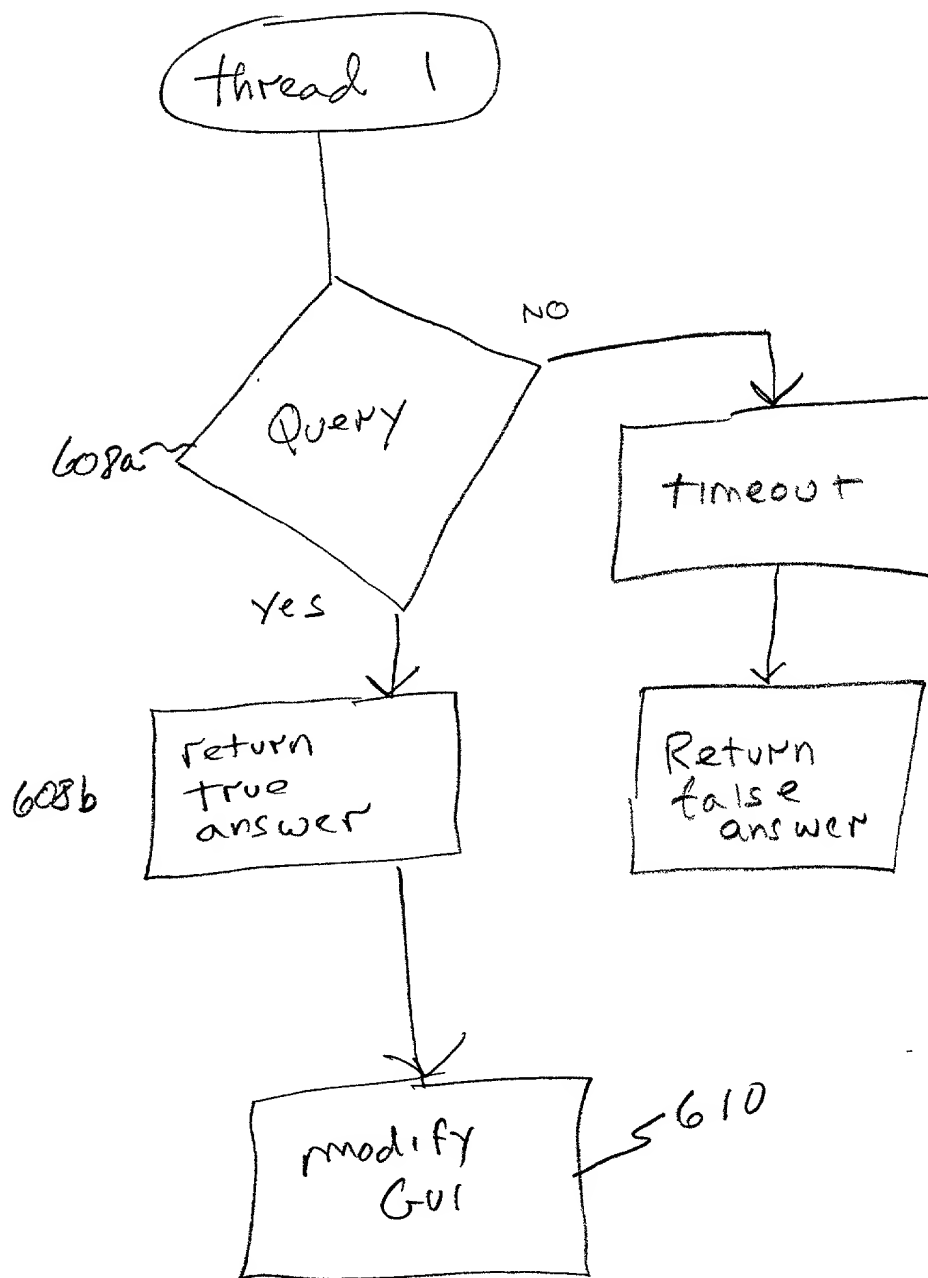


Fig. 7

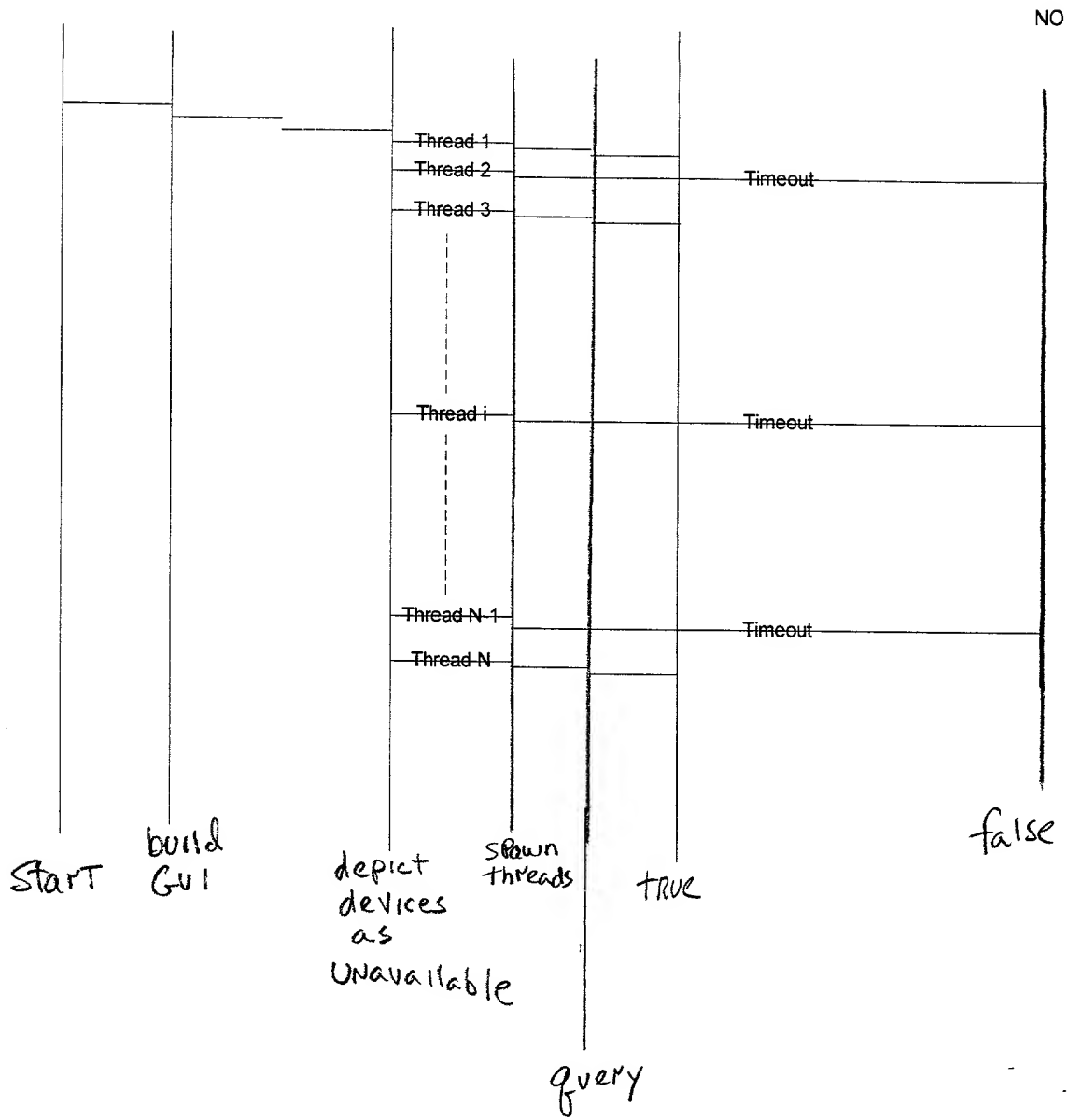


Fig. 8

Code:

```
// DoEnable() gets an array of GUI components, and will enable/disable the components based on their availability in the network.
void DoEnable(GUIComponent arGUI[], unsigned short sz)
{
    // Disable entire GUI
    for (unsigned short I=0;I<sz;I++)
        arGUI[I].DisableComponent ();

    for (I=0;I<sz;I++)
    {
        AfxBeginThread(ValidateRemoteHost, (LPVOID) &arGUI[I]);
    }
}
```

Fig. 9

```

struct GUIComponent
{
    // will return IP address associated with the network component that the current GUI element represents.
    unsigned long GetIPAddress ();
    void EnableComponent(); // display component as active
    void DisableComponent(); // display component as inactive.
};

// worker thread which will attempt a socket connection with remote host.
UINT ValidateRemoteHost( LPVOID pParam )
{
    GUIComponent * pGUIComponent =( GUIComponent *)pParam;

    if (QueryRemoteHost(pGUIComponent->GetIPAddress()) )
        pGUIComponent->EnableComponent();
    return 0;
}

```

Fig. 10

```

BOOL QueryRemoteHost(unsigned long lIpAdd)
{
    BOOL bRetVal=FALSE;
    // create new socket
    SOCKET S = socket( AF_INET, SOCK_STREAM, IPPROTO_IP);
    if ( INVALID_SOCKET == S)
        return FALSE;

    SOCKADDR_IN dest_sin;
    ZeroMemory(&dest_sin, sizeof(SOCKADDR_IN));
    dest_sin.sin_family = AF_INET;
    dest_sin.sin_port = htons(21);
    dest_sin.sin_addr.S_un.S_addr = lIpAdd;

    if (SOCKET_ERROR == connect(S,(PSOCKADDR)&dest_sin, sizeof( dest_sin)))
        bRetVal = FALSE;
    else
        bRetVal = TRUE;

    closesocket(S);
    return bRetVal;
}

```

Fig. 11

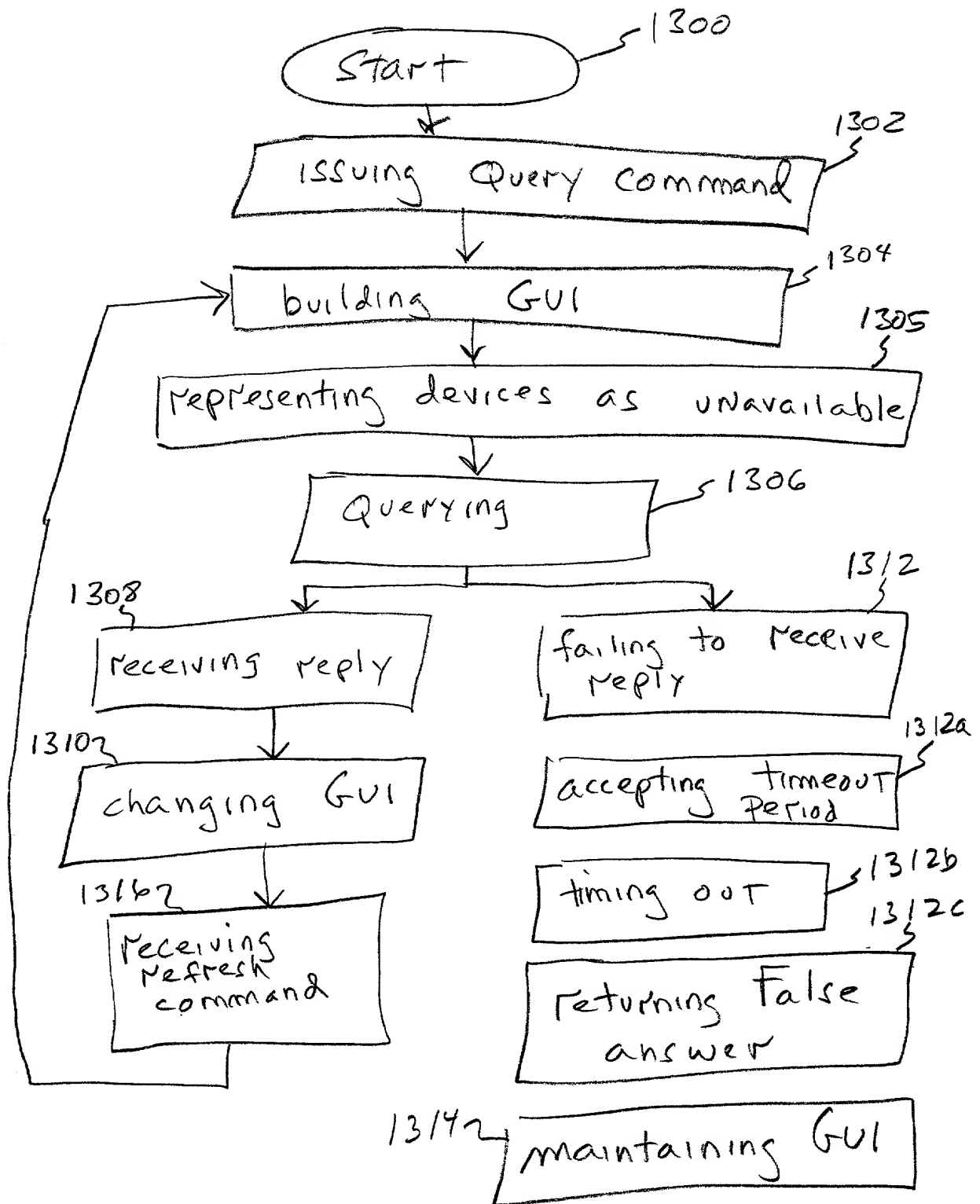


Fig. 13

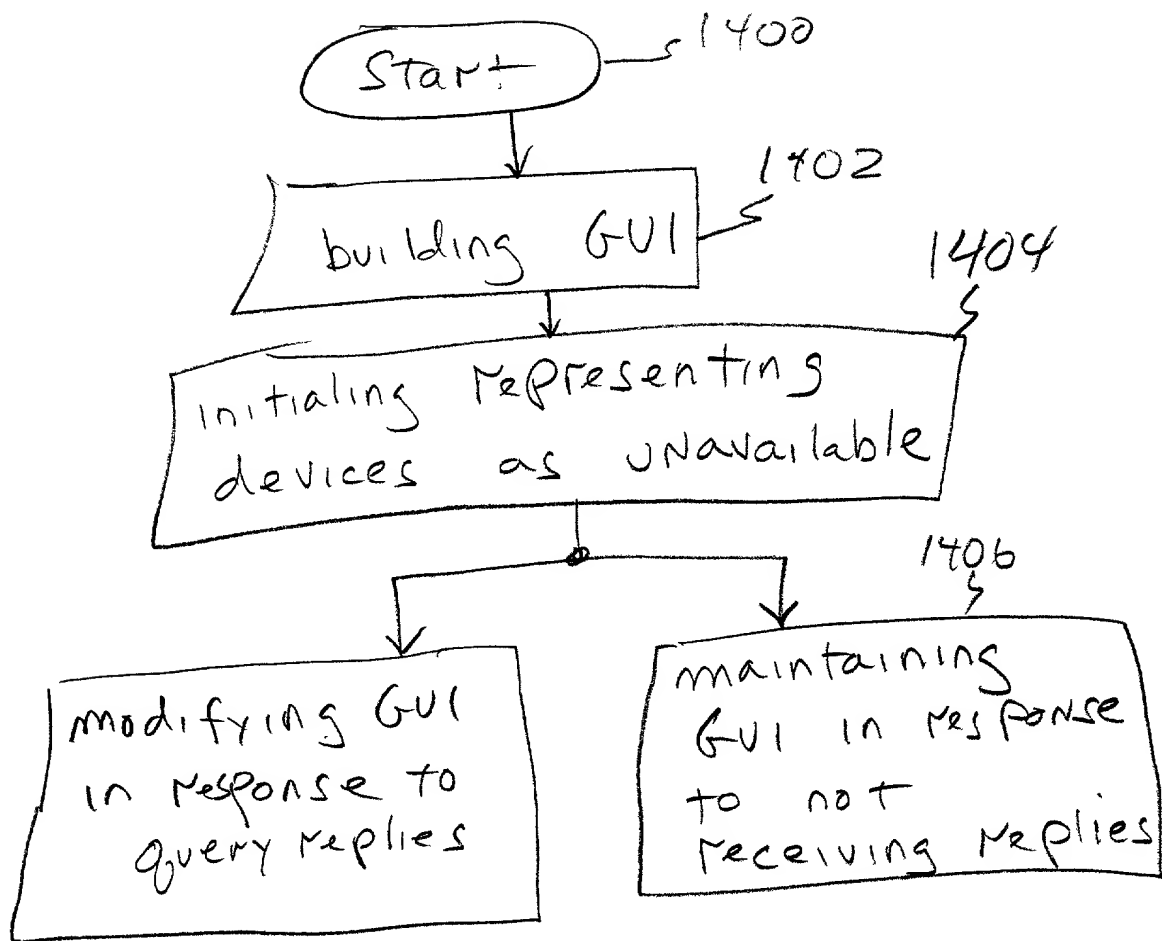


Fig. 14